# 4주차

## 배열과 포인터

## 문자열

## 다차원 배열, 다중 포인터

# 조건문

## if else 문

```
if(조건식)
{
    조건이 참일때
}
else
{
    조건이
    거짓일때
}
```

```
if(조건식)
{
    조건이 참일때
}
else if(조건식)
{
  이전 조건이 거짓
  현재 조건이 참일때
}
```

## switch case 문

```
switch(변수)
{
case 값 : 해당 코드
          (break;)
case 값 : 해당 코드
          (break;)
default :
        어떤case에도
        해당 안될때
        break;
}
```

# 반복문

## while 문

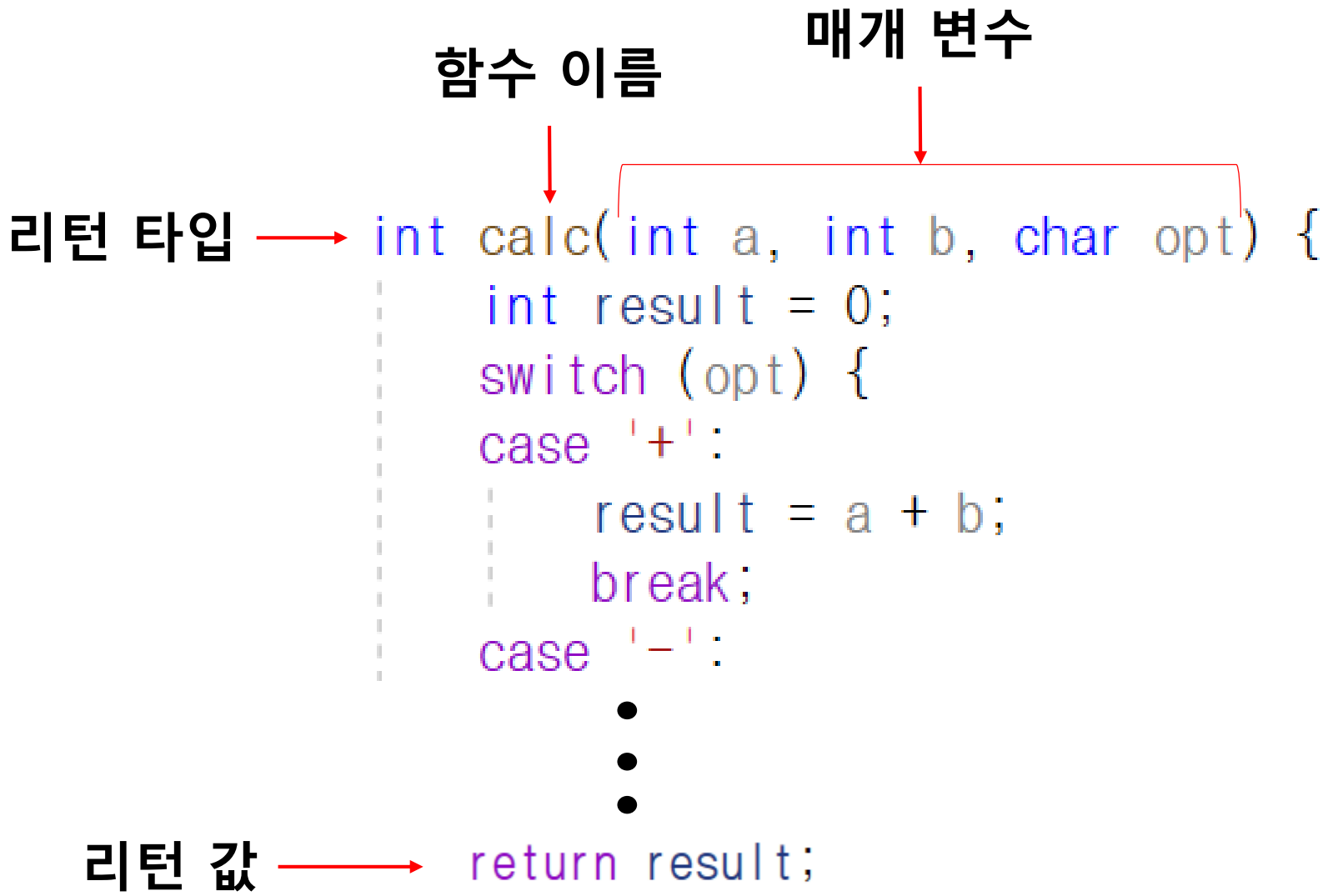
```
while(조건식)
{
    반복실행 코드
}
```

```
do
{
    반복실행 코드
} while(조건식)
```

## for 문

```
for(초기값 ; 조건식 ; 증감식)
{
    반복실행 코드
}
```

# 함수


매개 변수
함수 이름
리턴 타입
int calc(int a, int b, char opt) {
int result = 0;
switch (opt) {
case '+':
result = a + b;
break;
case '-':
리턴 값
return result;

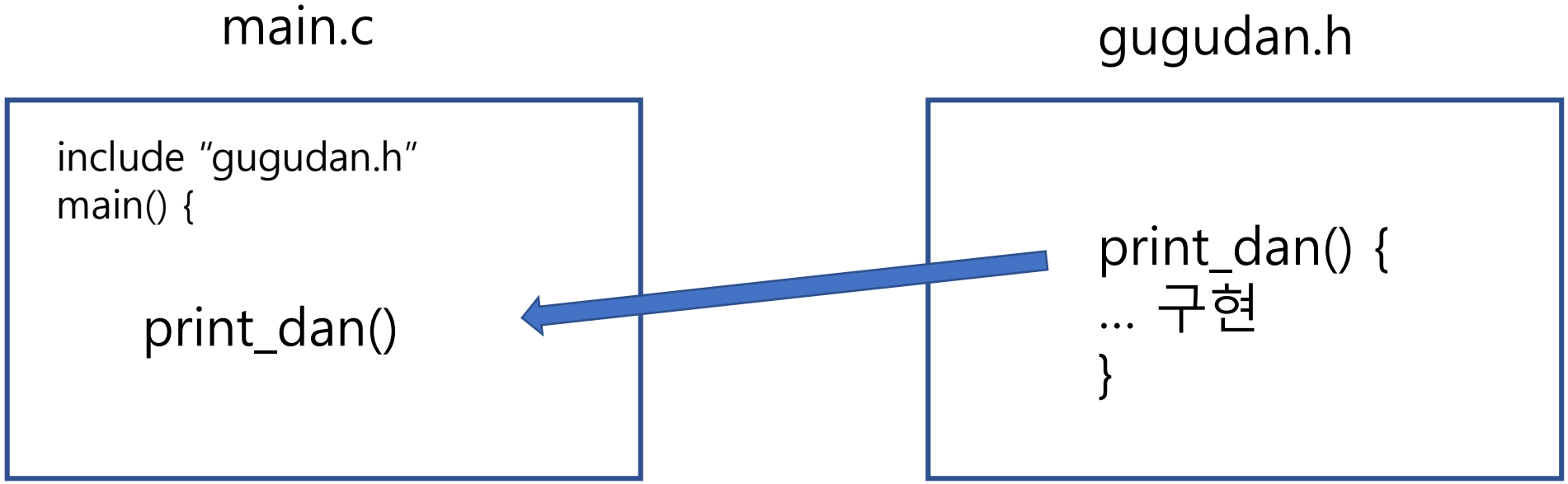
# 특정 구구단을 출력하는
# 함수 print_dan 구현

```
void print_dan(int n) {
    if (n < 2 || n > 9) {
        printf("값이 잘못 입력되었습니다.₩n");
        return;
    }

    for (int i = 1; i < 10; i++) {
        printf("%d x %d = %d₩n", n, i, n * i);
    }
}
```

# print_dan 함수 선언과 함수 사용

```c
void print_dan(int n);

int main(void) {
    int num;
    scanf_s("%d", &num);
    print_dan(num);
}
```

# 헤더파일과 include



# 헤더파일과 include

```c
#include <stdio.h>
#include "gugudan.h"

int main(void) {
    int num;
    puts("출력할 단을 입력하세요");
    scanf_s("%d", &num);
    print_dan(num);
}
```

# 헤더파일 중복과
# #pragma once

```
#include “gugudan.h”
#include “dan.h”
main() {
        print_dan()
        dan()
```

main.c

```
print_dan() {
… 구현
}
```

gugudan.h

```
#include “gugudan.h”
dan() {
     print_dan()
}
```

dan.h

# 헤더파일 중복과 #pragma once

```
#include “gugudan.h”
#include “dan.h”
main() {
        print_dan()
        dan()
```

main.c

```
#pragma once
print_dan() {
… 구현
}
```

gugudan.h

```
#include “gugudan.h”
dan() {
     print_dan()
}
```

dan.h

# 헤더파일 과 소스파일

```
print_dan()
```

gugudan.h

```
#include “gugudan.h”
#include “dan.h”
main() {
        print_dan()
        dan()
```

main.c

```
#include “gugudan.h”
dan() {
     print_dan()
}
```

dan.h

```
print_dan() {
… 구현
}
```

print_gugu.c

# 헤더파일 통합

```
#include “gugudan.h”
main() {
    print_dan()
    dan()
```

main.c

```
print_dan()
dan() {
    print_dan()
}
```

gugudan.h

```
print_dan() {
… 구현
}
```

print_gugu.c

# 포인터(pointer)

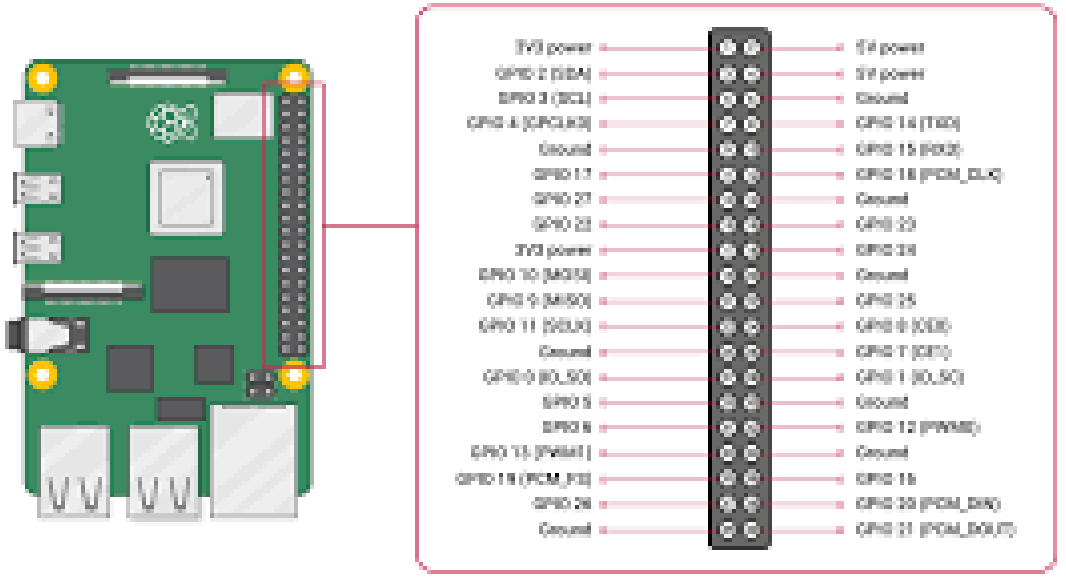
## : 메모리 주소값을 저장,

**자료형과 관계없이**
**4byte크기(visual studio)**

```
int a = 10;
char c = 'C';

int* p_a = &a;
char* p_c = &c;
```

# Why 포인터 씀?

## 1. 하드웨어 직접제어

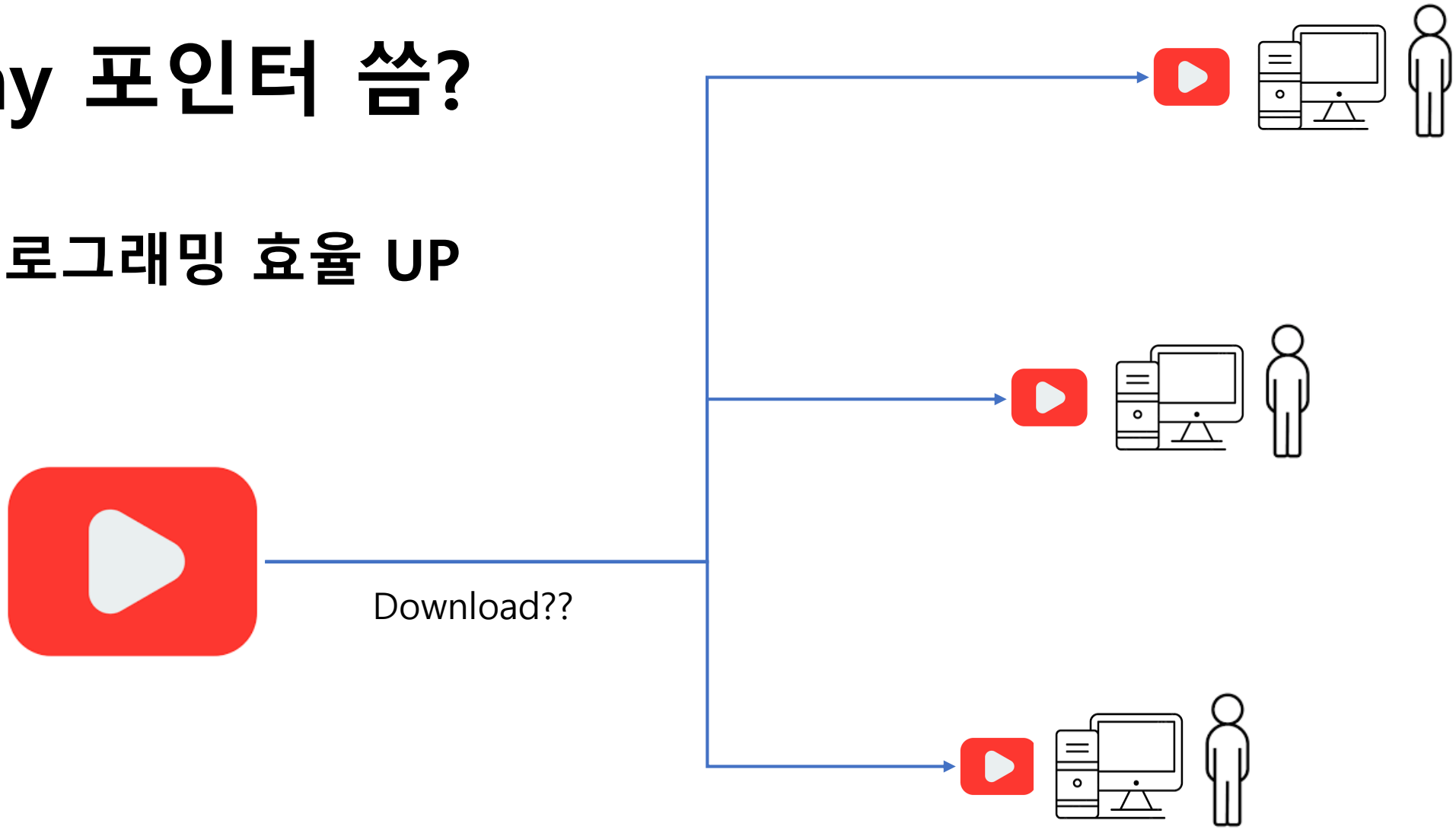


GPIO(General Purpose Input Output)

```
#define GPIO_BASE       0x00006F80

#define GPACRL          ((volatile unsigned long*)(GPIO_BASE + 0x00000000))
#define GPAQSEL1        ((volatile unsigned long*)(GPIO_BASE + 0x00000002))
#define GPAQSEL2        ((volatile unsigned long*)(GPIO_BASE + 0x00000004))
#define GPAMUX1         ((volatile unsigned long*)(GPIO_BASE + 0x00000006))
#define GPAMUX2         ((volatile unsigned long*)(GPIO_BASE + 0x00000008))
#define GPADIR          ((volatile unsigned long*)(GPIO_BASE + 0x0000000A))
#define GPAPUD          ((volatile unsigned long*)(GPIO_BASE + 0x0000000C))
```

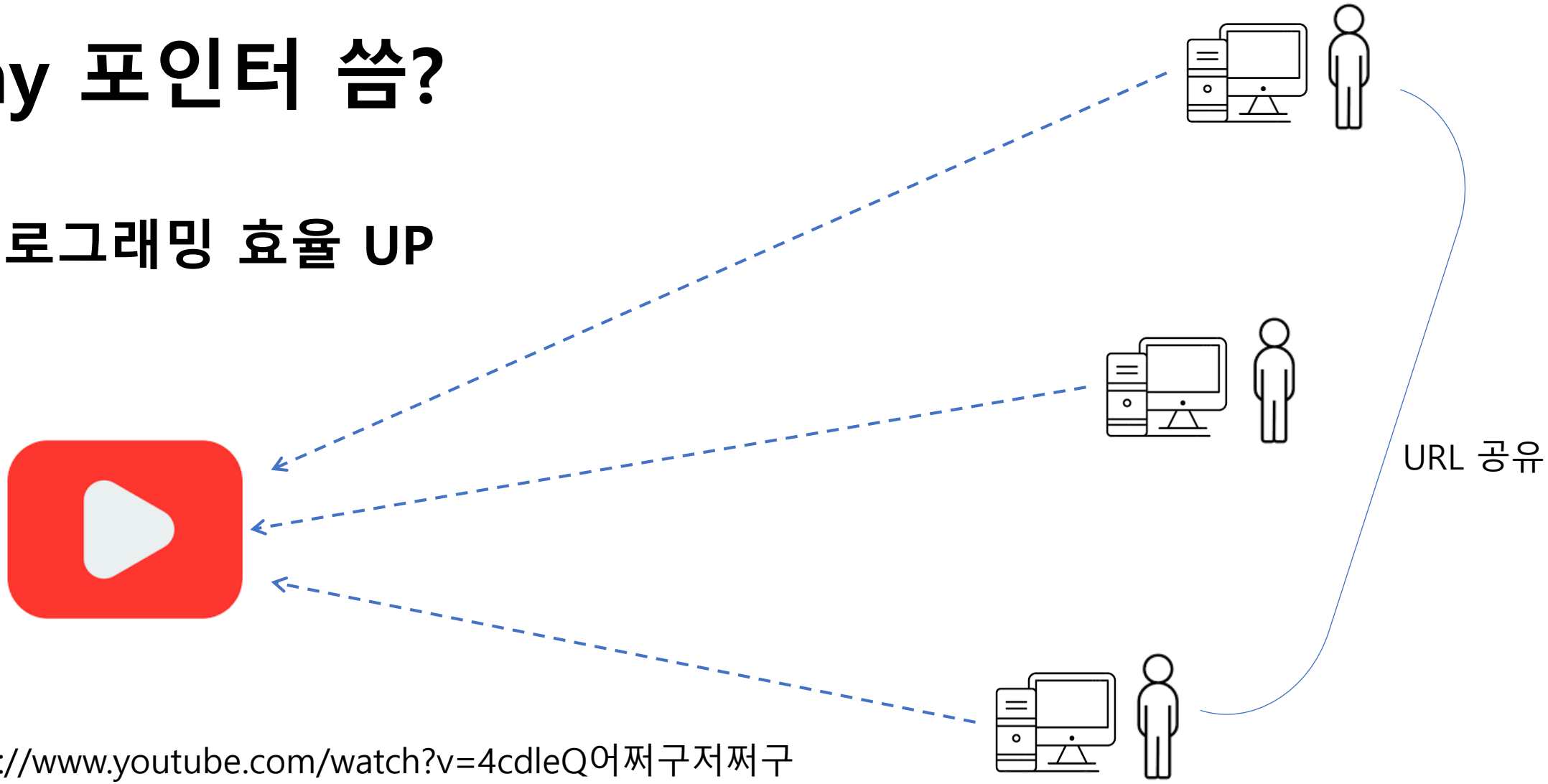
# Why 포인터 씀?

## 2. 프로그래밍 효율 UP



# Why 포인터 씀?

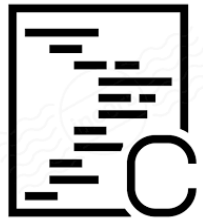
## 2. 프로그래밍 효율 UP



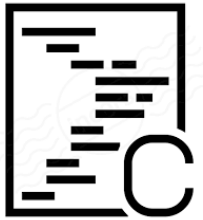

URL주소
예) https://www.youtube.com/watch?v=4cdleQ어쩌구저쩌구

# Why 포인터 씀?

## 2. 프로그래밍 효율 UP





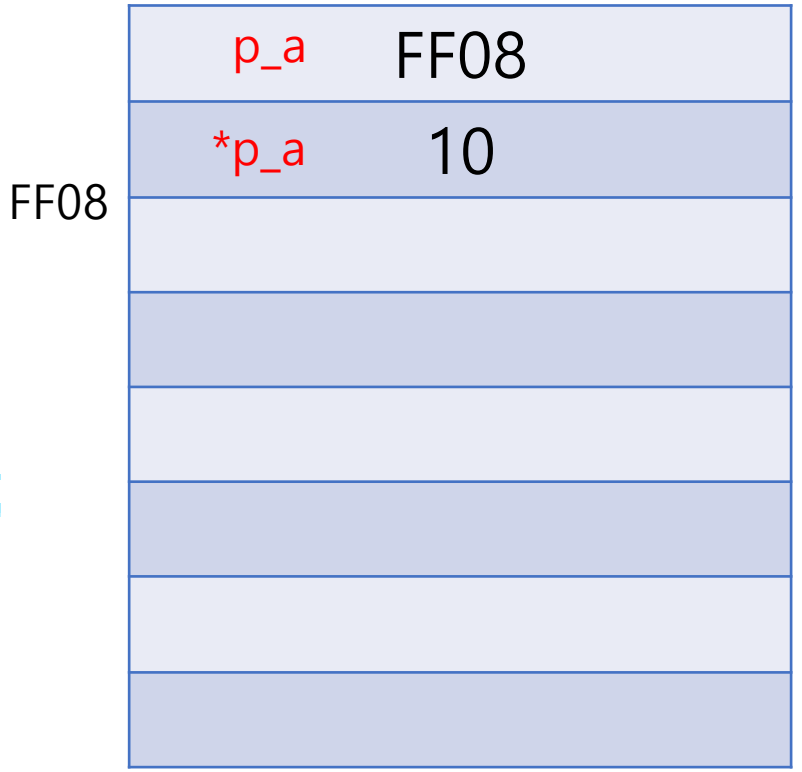

포인터
p 공유

메모리 주소
예) p = 0057FE9C

# 주소와 포인터 참조 (Reference)

```c
int main(void) {
    int a = 10;
    int* p_a = &a;

    printf("&a : %p, a : %d₩n", &a, a);
    printf("&p_a : %p, p_a: %p₩n", &p_a, p_a);
    printf("*p_a : %d₩n", *p_a);
    *p_a = 5;
    printf("대입 후 a : %d₩n", a);
}
```

**&** Ampersand 앰퍼샌드
- 주소 연산자
(address of- operator)



# * (별표, asterisk 에스터리스크) 의 용도

```
int n = 10;
int* pointer;
pointer = &n;
pointer = (int*)0x002FC240;
*pointer = 100;
n *= 10;
```

1. 자료형 포인터 선언

2. 자료형 포인터 캐스팅

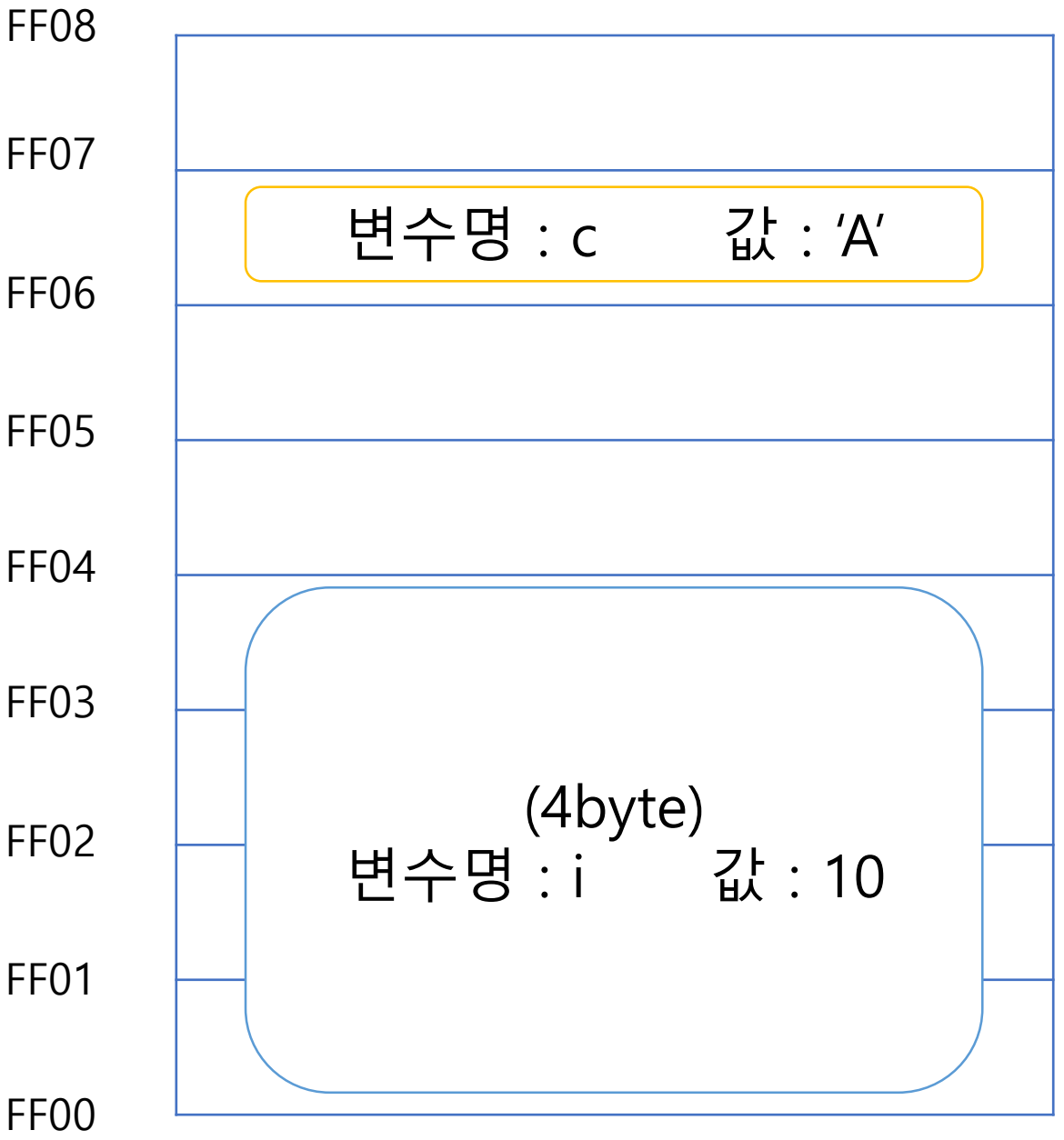
3. 간접 참조 연산자
(indirect reference operator)
포인터 주소안에 들어있는 값

4. 곱셈..

# 기본 자료형

int, short, char, float …

```
int main(void) {
    char c = 'A';
    int i = 10;
```



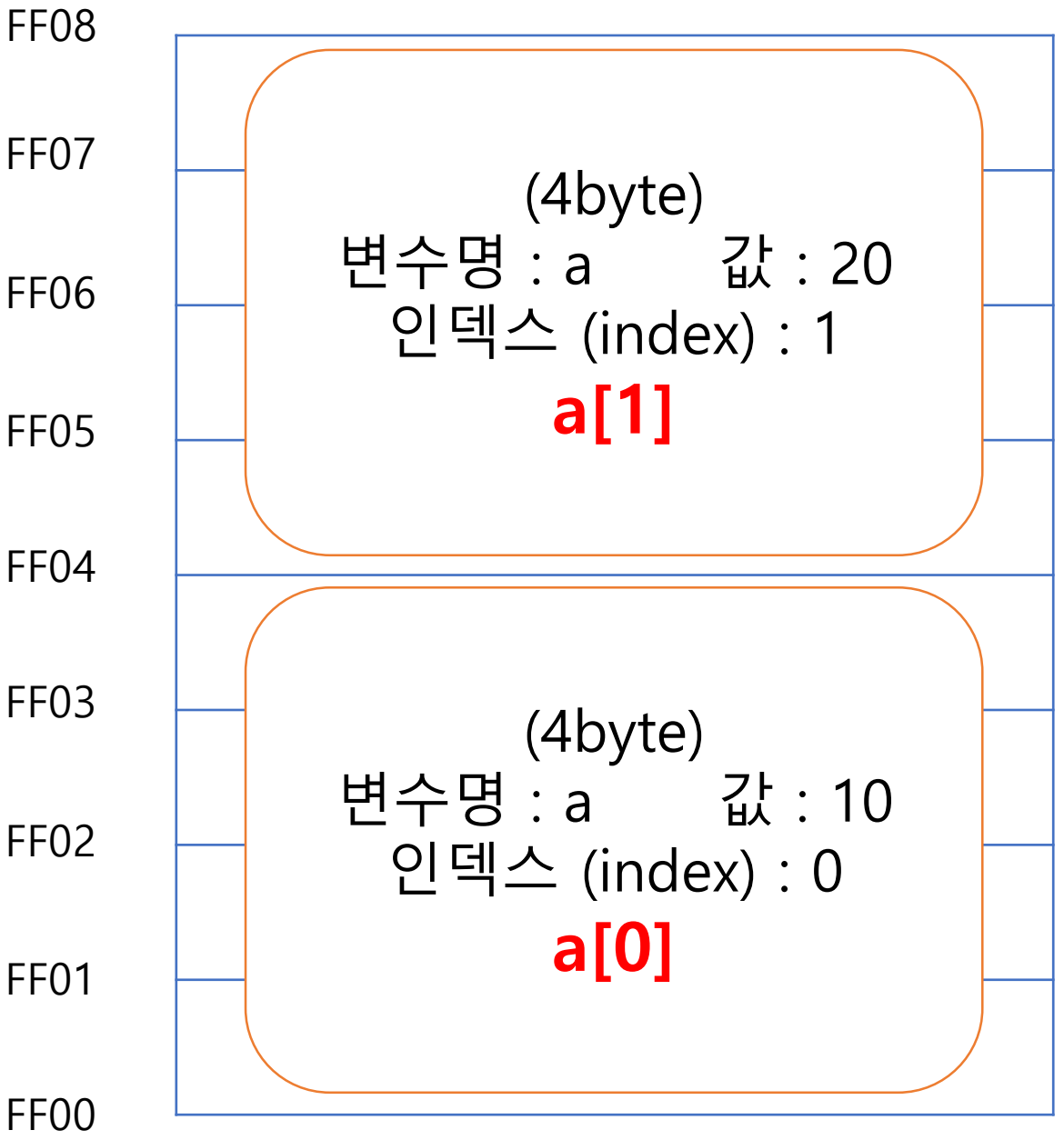
# 배열 (array)

: 연속된 자료형

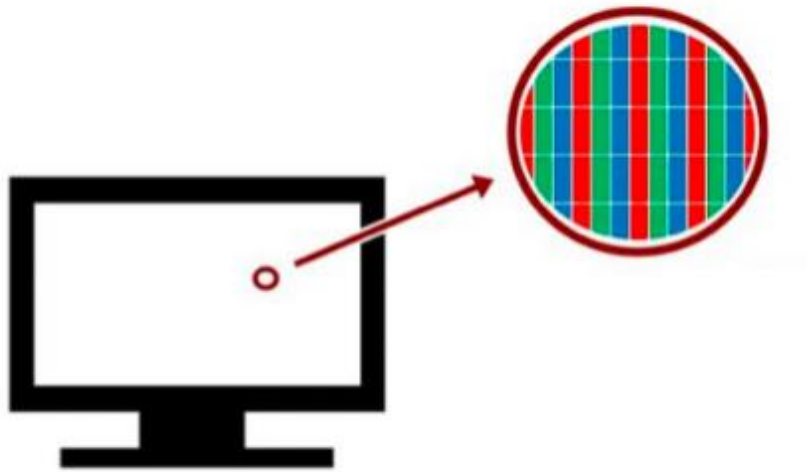
```
int a[2] = { 10, 20 };
```

a[0] a[1]

배열 크기
(원소의 개수)



# Why 배열 필요?

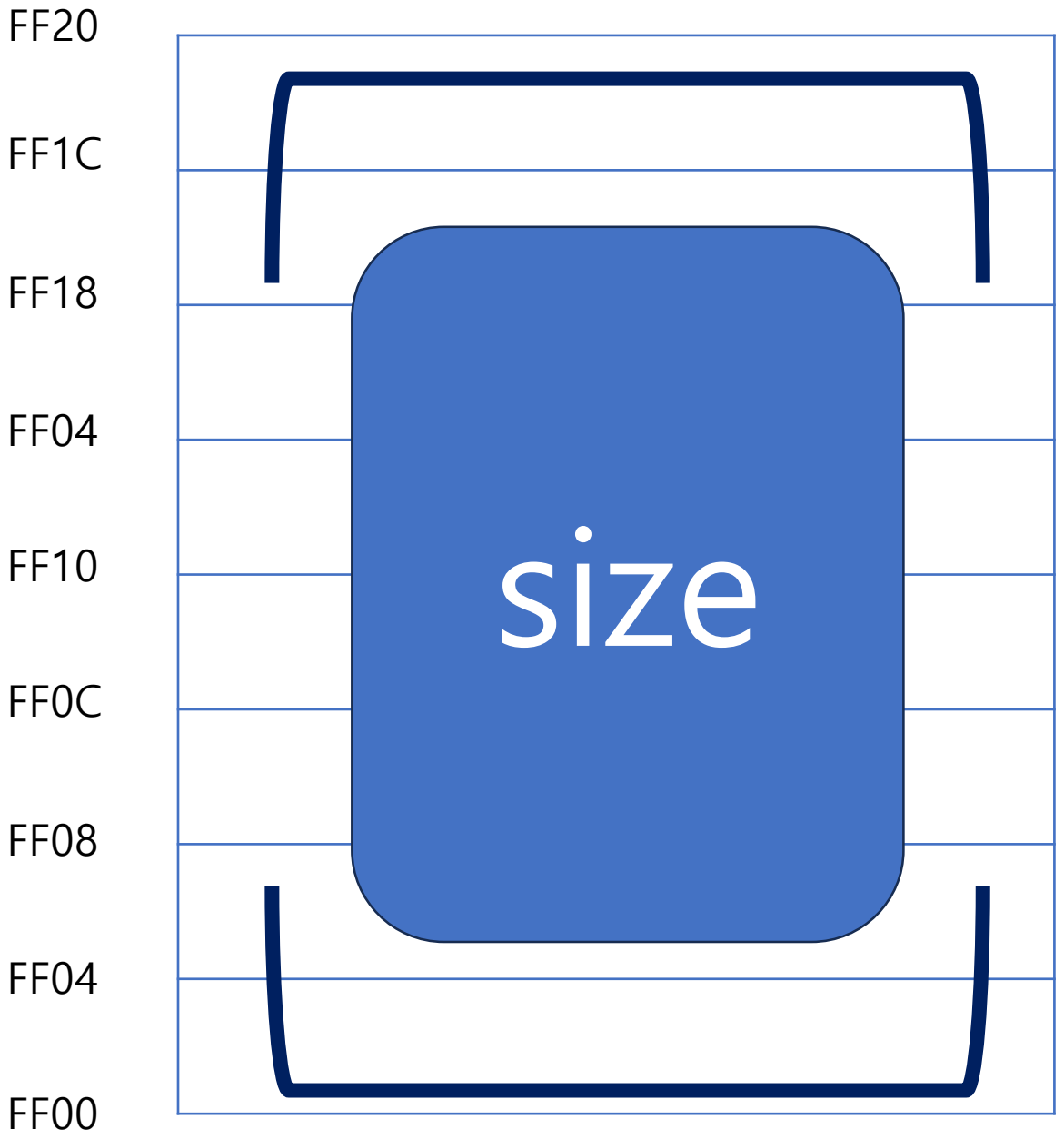


$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{11} & a_{12} & \cdots & a_{1n} \end{bmatrix}, \; a_{ij} \in R$$

# 배열 선언 형태

필수 : 크기지정

```c
#include <stdio.h>
#define MAX_SIZE 100
int main(void) {
    char a1[] = {'s','u','n','₩0'};
    float a2[2] = { 1.41, 0.6 };
    int a3[100];
    double a4[MAX_SIZE];
    //int a5[]; // 허용X
```



# 배열과 for문

```c
int main(void) {
    int odd[] = { 1, 3, 5, 7, 9 };

    for (int i = 0; i < 5; i++) {
        printf("odd[%d] : %d₩n", i, odd[i]);
    }
}
```

# for문 초기화

```c
int main(void) {
    int odd[] = { 1, 3, 5, 7, 9 };
    int even[5];

    for (int i = 0; i < 5; i++) {
        printf("odd[%d] : %d₩n", i, odd[i]);
        // even의 원소를 배열odd를 이용하여 초기화
        // even[] = { 2, 4, 6, 8, 10 }
        even[i] = odd[i] + 1;
        printf("even[%d] : %d₩n", i, even[i]);
    }
}
```

```c
int main(void) {
    int odd[] = { 1, 3, 5, 7, 9 };
    int even[5];
    int sum[10];
    //sum[]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Microsoft Visual Studio 디버그 콘솔

```
sum[0]: 1
sum[1]: 2
sum[2]: 3
sum[3]: 4
sum[4]: 5
sum[5]: 6
sum[6]: 7
sum[7]: 8
sum[8]: 9
sum[9]: 10
```

```c
}
```

# 배열 요소 개수 구하기

```
int count = sizeof(odd) / sizeof(int);
printf("sizeof(odd) : %d, sizeof(int) : %d₩n"
    "count : %d₩n", sizeof(odd), sizeof(int), count);
```

# 배열 초기값과 쓰레기값

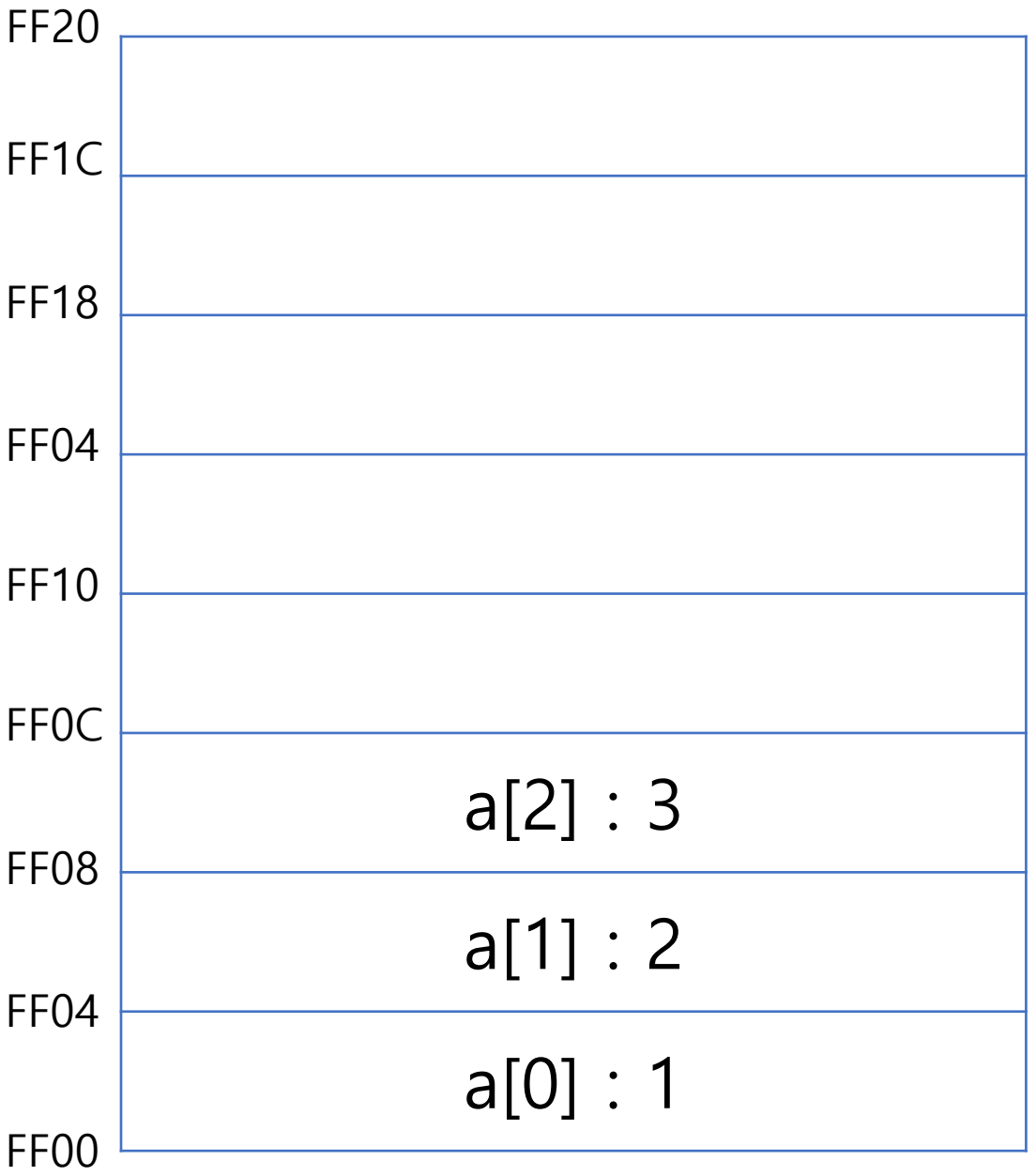
```
int main(void) {
    int a[8] = { 1, 2, 3, 4, 5 };
    int b[8];

    for (int i = 0; i < 5; i++) {
        b[i] = i;
    }

    printf("a[6] : %d₩n", a[6]);
    printf("b[6] : %d₩n", b[6]);
    printf("a[20] : %d₩n", a[20]);
    printf("b[20] : %d₩n", b[20]);
}
```

| |
|---|
| a[7] / b[7] |
| a[6] / b[6] |
| a[5] / b[5] |
| a[4] : 5 / b[4] : 5 |
| a[3] : 4 / b[3] : 4 |
| a[2] : 3 / b[2] : 3 |
| a[1] : 2 / b[1] : 2 |
| a[0] : 1 / b[0] : 1 |

# 배열 이름

- 첫번째 원소의 주소
- 전체 배열의 크기정보

```
int main(void) {
    int a[] = { 1, 2, 3 };
    printf("%p₩n", a);
    printf("%p₩n", &(a[0]));
    printf("%d₩n", sizeof(a));
}
```

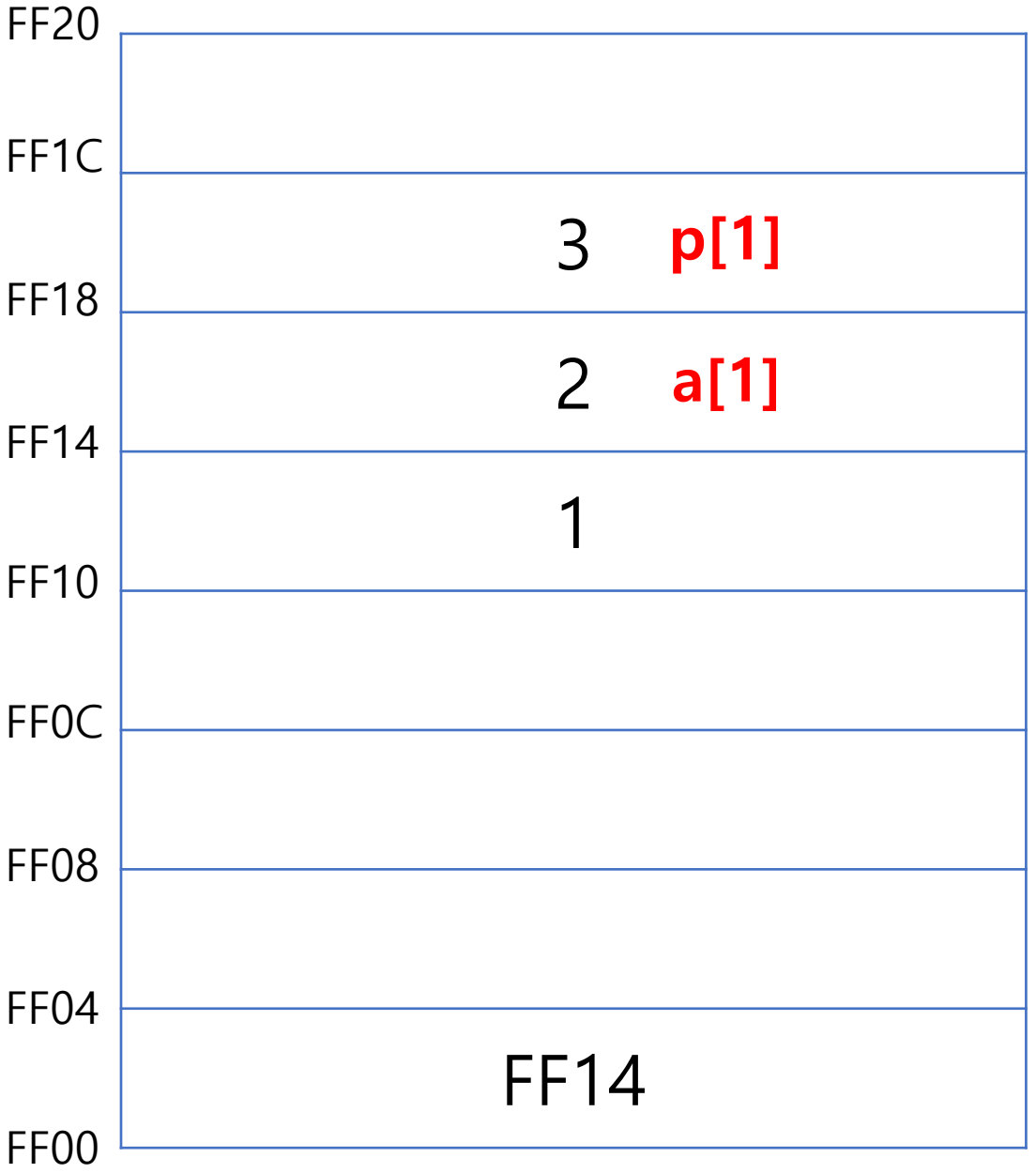


# 배열과 포인터
## 관계

```
int a[] = { 1, 2, 3 };
int* p = a;
printf("배열 a[2] : %d₩n", a[2]);
printf("포인터 p[2] : %d₩n", p[2]);
printf("배열 a의 크기 : %d₩n", sizeof(a));
printf("포인터 p의 크기 : %d₩n", sizeof(p));
```

# 배열과 포인터
## 관계

```
int a[] = { 1, 2, 3 };
int* p = &a[1];
printf("a[1] : %d₩n", a[1]);
printf("p[1] : %d₩n", p[1]);
```



# 배열 이름 특징

:첫번째 원소 주소값과 배열크기 정보 ,수정 불가

```
int* a[] = { 1, 2, 3 };
int* b[] = { 4, 5, 6 };
int* p = a;
printf("sizeof(a) : %d₩n", sizeof(a)); // 배열크기 정보
printf("&a[0] : %p₩n", &a[0]);
printf("a : %p₩n",a); // 첫번째 원소의 주소

//a = b;  재할당 불가능
p = b; // 포인터는 가능
printf("*p : %d₩n", *p);
```

# 배열 요소는
# :일반 변수처럼 (수정가능)

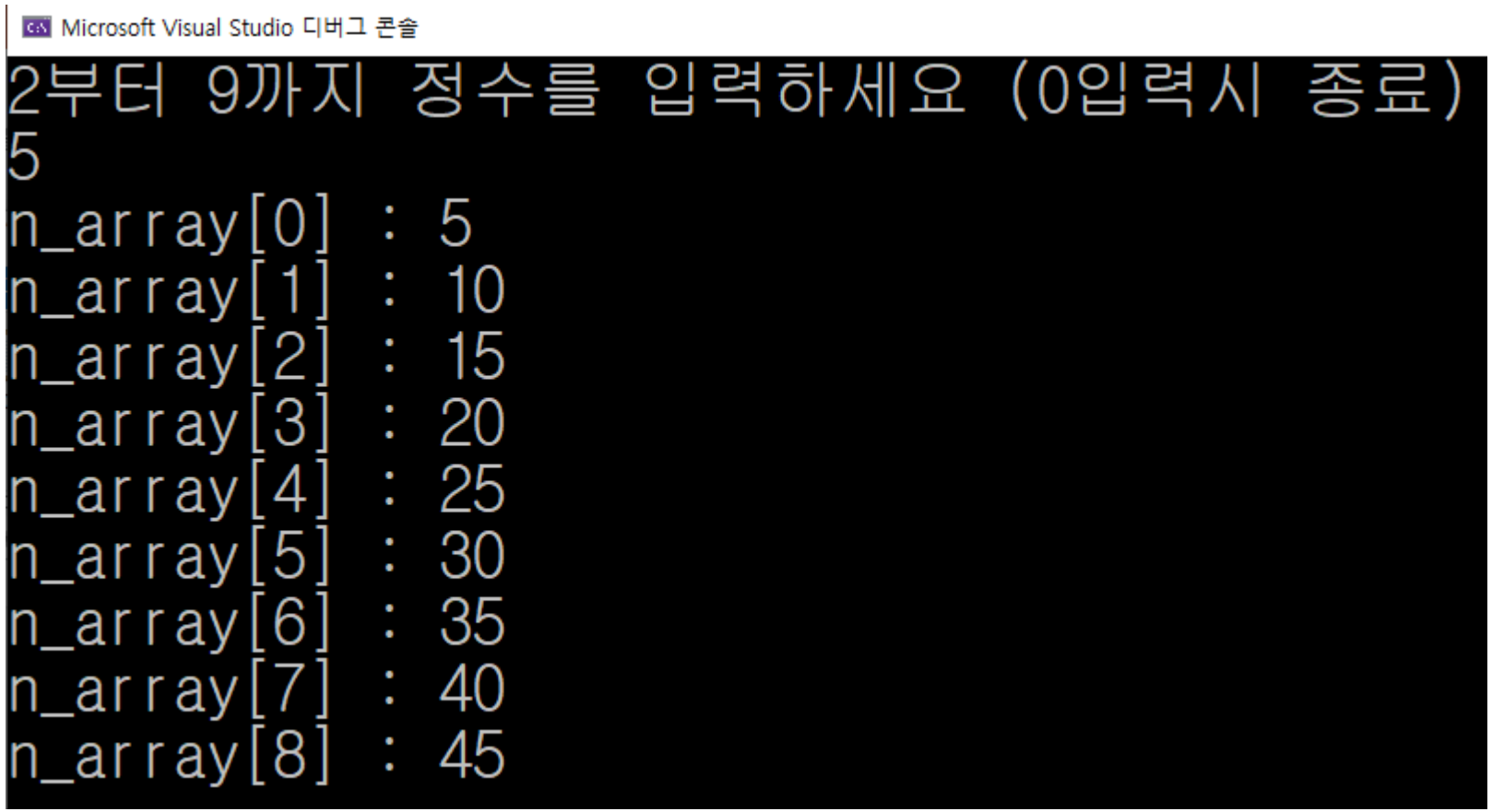
```
int main(void) {
    int a[] = { 1, 2, 3 };
    a[0] = 4;
    a[1] = 5;
    a[2] = 6;
}
```

# 함수 dan 호출 수정

```
void dan(int n_dan, int n_array[]);

int main(void) {
    int n_dan;
    int n_array[9];
    puts("2부터 9까지 정수를 입력하세요 (0입력시 종료)");
    scanf_s("%d", &n_dan);
    dan(n_dan, n_array);
    for (int i = 0; i < 9; i++) {
        printf("n_array[%d] : %d₩n", i, n_array[i]);
    }
}
```

# 함수 dan 재정의

```
void dan(int n_dan, int n_array[]) {
}
```



# 배열과 포인터 주의점

```c
int main(void) {
    int a[] = { 1, 2, 3, 4 };
    int* p = a;
    // int* p = { 1, 2, 3, 4 };
    // 이런식의 할당은 안됨.

    int t;
    int* e = &t;
    // 위험한 코드
    t = a[10];
    t = e[10];

    // 심각하게 위험한 코드들
    a[20] = 10;
    e[20] = 20;
```


FF05

FF04

FF03

FF02


# 포인터 연산, NULL값

```
int a[] = { 1, 2, 3, 4 };
int* p = NULL;

printf("a : %p₩n", a);
printf("*a : %d₩n", *a);

if (p == NULL) { // p == 0 과 같다
    p = a;
}

printf("p : %p₩n", p);
printf("p + 1 : %p₩n", p + 1);
printf("++p : %p₩n", ++p); // a는 증감연산 불가
printf("--p : %p₩n", --p);
```

# 포인터 연산들

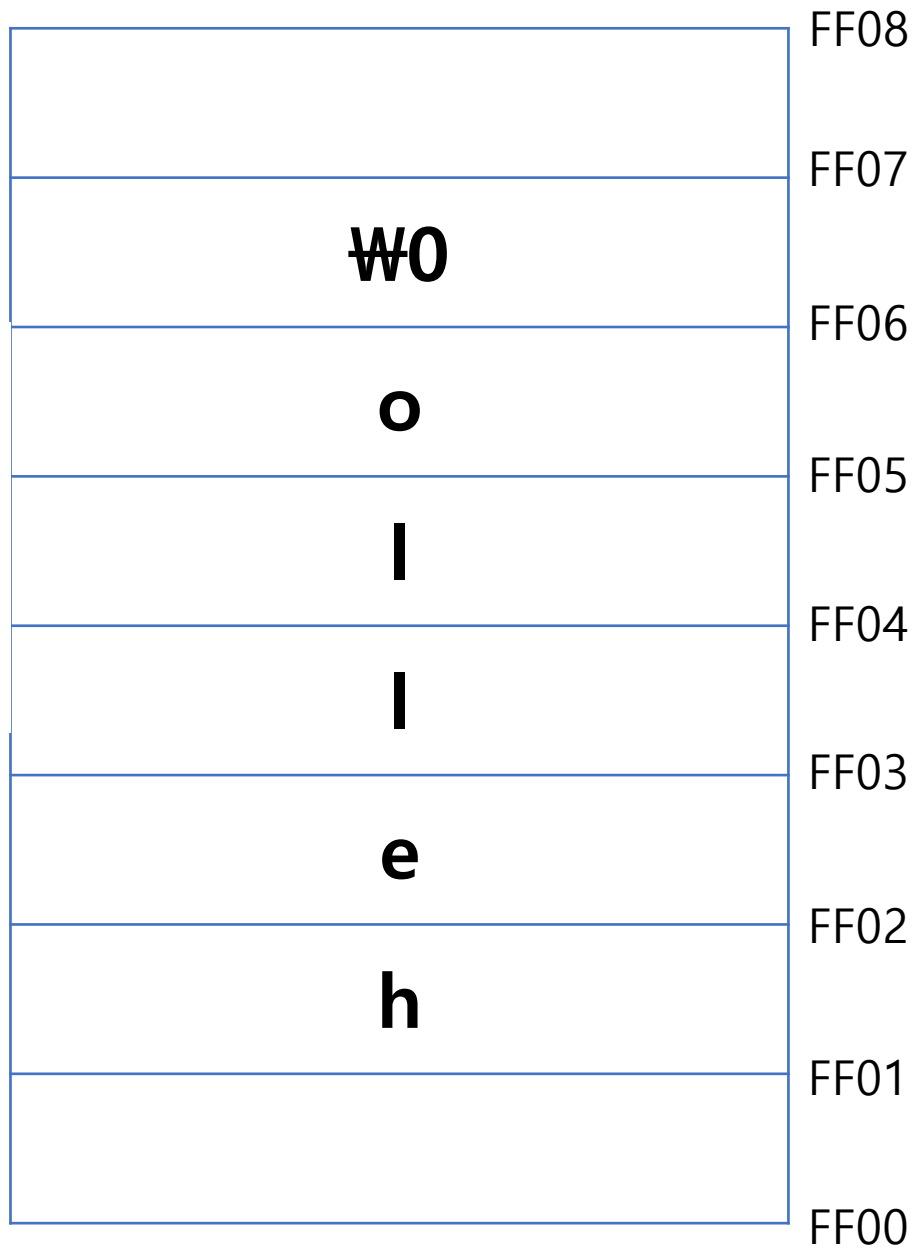
```
printf("a[1] : %d, p[2] : %d₩n", a[1], p[2]);
printf("&a[1] : %p, &p[2] : %p₩n", &a[1], &p[2]);

printf("&a[1] > &p[2] : %d₩n", &a[1] > &p[2]);
printf("a + 1 != p + 2 : %d₩n", a + 1 != p + 2);

printf("&p[2] - &a[1] : %d₩n", &p[2] - &a[1]);
printf("p + 2 - a + 1 : %d₩n", p + 2 - p + 1);
printf("(p + 2) - (a + 1) : %d₩n", (p + 2) - (p + 1));
```

# 포인터 연산시 1의 크기

```
int main(void) {
    char c[10];
    short sh[10];
    int i[10];
    double d[10];

    printf("c: %d %d₩n", c, c + 1);
    printf("sh: %d %d₩n", sh, sh + 1);
    printf("i: %d %d₩n", i, i + 1);
    printf("d: %d %d₩n", d, d + 1);
}
```

# 문자형 배열

```c
int main(void) {
    char a[6] = { 'h','e','l','l','o','₩0' };
    char str[] = "hello";
    printf("%s₩n", a);
}
```



# '₩0' (NUL문자)

```c
int main(void) {
    char a[6] = { 'h','e','l','l','o','₩0' };
    char str[] = "hello";
    printf("%s₩n", a);

    a[3] = '₩0';
    printf("%s₩n", a);

    a[1] = 0;
    printf("%s₩n", a);
}
```

| 10진 | 16진 | 문자 |
|---|---|---|
| 0 | 0x00 | NUL |
| 48 | 0x30 | 0 |

# 배열로 성적관리

## -지우지마시고 주석처리

```c
#include <stdio.h>
#define MAX_NAME 100
#define SCORES 3

int main(void) {
    char name[MAX_NAME];
    int scores[SCORES];
    printf("학생이름 : ");
    scanf_s("%s", name, MAX_NAME);
    printf("국 영 수 형태로 입력: ₩n");
    scanf_s("%d %d %d", scores, scores + 1, &scores[2]);

    printf("%s 학생의 점수는 (국 영 수): ₩n", name);
    for (int i = 0; i < SCORES; i++) {
        printf("%d ", scores[i]);
    }
}
```

# 점수 합계 계산 함수 만들기

```
#define MAX_NAME 100
#define SCORES 3
int total_score(int* scores);
```

●
●
●

```
printf("%s 학생의 점수는 (국 영 수): ₩n", name);
for (int i = 0; i < SCORES; i++) {
    printf("%d ", scores[i]);
}
printf("점수 합계는 %d점 입니다.₩n", total_score(scores));
```

# 문자형 배열과 문자열 포인터

```c
int main(void) {

    char a[] = "array";
    char* p_str = "pointer";

}
```

# 배열과 포인터 비교

1. 배열명으로 재할당이 안됨
   포인터는 가능

```
int main(void) {
    char a[] = "array";
    char* p_str = "pointer";

    a = "fail";
    p_str = "ok"
}
```

# 배열과 포인터 비교

2. 배열명으로 전체 배열크기를 알수있음, 포인터는 무조건 4byte

*string.h 의 strlen()함수

: 널문자 제외 오직 문자열의 크기

```c
#include <string.h>

int main(void) {
    char a[] = "array";
    char* p_str = "pointer";

    printf("%d₩n", sizeof(a));
    printf("%d₩n", strlen(a));
    printf("%d₩n", sizeof(p_str));
    printf("%d₩n", strlen(p_str));
}
```

# strlen 이용 예외처리

```
int main(void) {
    char pw[100];
```

Microsoft Visual Studio 디버그 콘솔

```
패스워드를 설정합니다 (8글자 이상 12글자 이하)
123123123
패스워드 123123123로 설정되었습니다.
```

Microsoft Visual Studio 디버그 콘솔

```
패스워드를 설정합니다 (8글자 이상 12글자 이하)
123
패스워드 길이가 잘못됐습니다.
}
```

# 배열과 포인터 비교

3. 인덱스로 원소에 접근 가능

(1) 인덱스는 0부터 시작

(2) 포인터 연산과 인덱스는 동일

ex) a[0] = *a, a[2] = *(a+2)

```c
int main(void) {
    char a[] = "array";
    char* p_str = "pointer";

    printf("%c₩n", a[3]);
    printf("%c₩n", p_str[2]);
}
```

# 배열과 포인터 비교

4. 인덱스로 원소 수정 가능

단 문자열 포인터로 생성한

문자열 원소는 수정불가

```c
int main(void) {
    char a[] = "array";
    char* p_a = a;
    char* p_str = "pointer";

    a[0] = 's';
    p_a[1] = 'p';
    printf("%s₩n", a);

    p_str[0] = 'w';
    printf("%s₩n", p_str);
}
```

# 배열과 포인터 하나만?

자료에 따라 배열과 포인터 둘다 필요함

| | | | | |
|---|---|---|---|---|
| P | A | R | K | |
| M | E | S | S | I |
| S | O | N | | |

**배열만 사용하면**

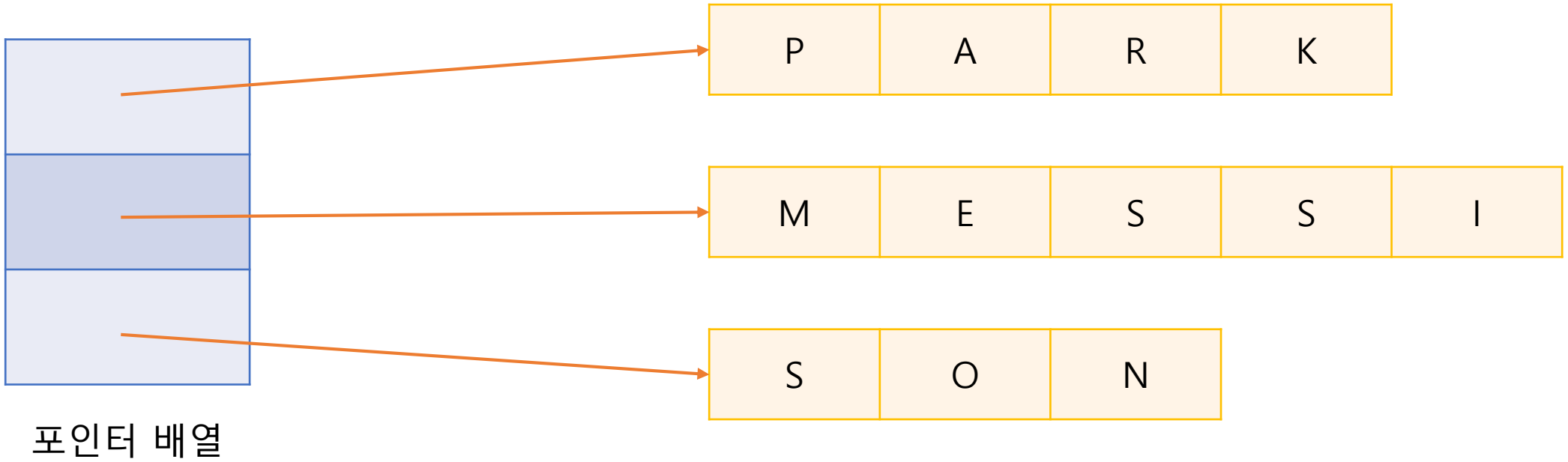
**낭비되는 공간 발생**

# 배열과 포인터 하나만?

포인터가 들어있는 배열로 처리함

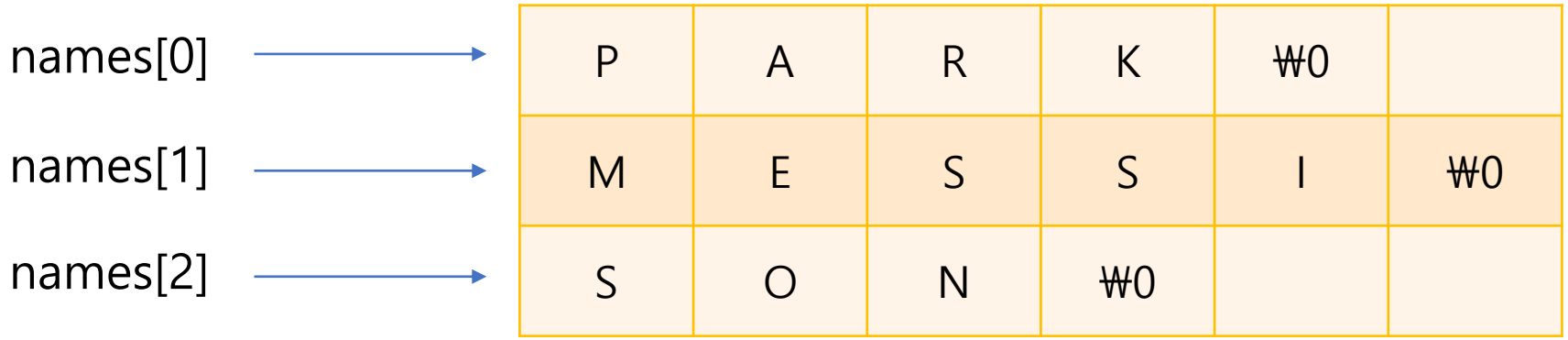
각각 필요한만큼만 메모리를 할당

=> 동적메모리 할당



# 다차원 배열 (2차원)

```
int main(void) {

    char names[3][6] = { "PARK", "MESSI", "SON" };

}
```



# 다차원 배열 (3차원)

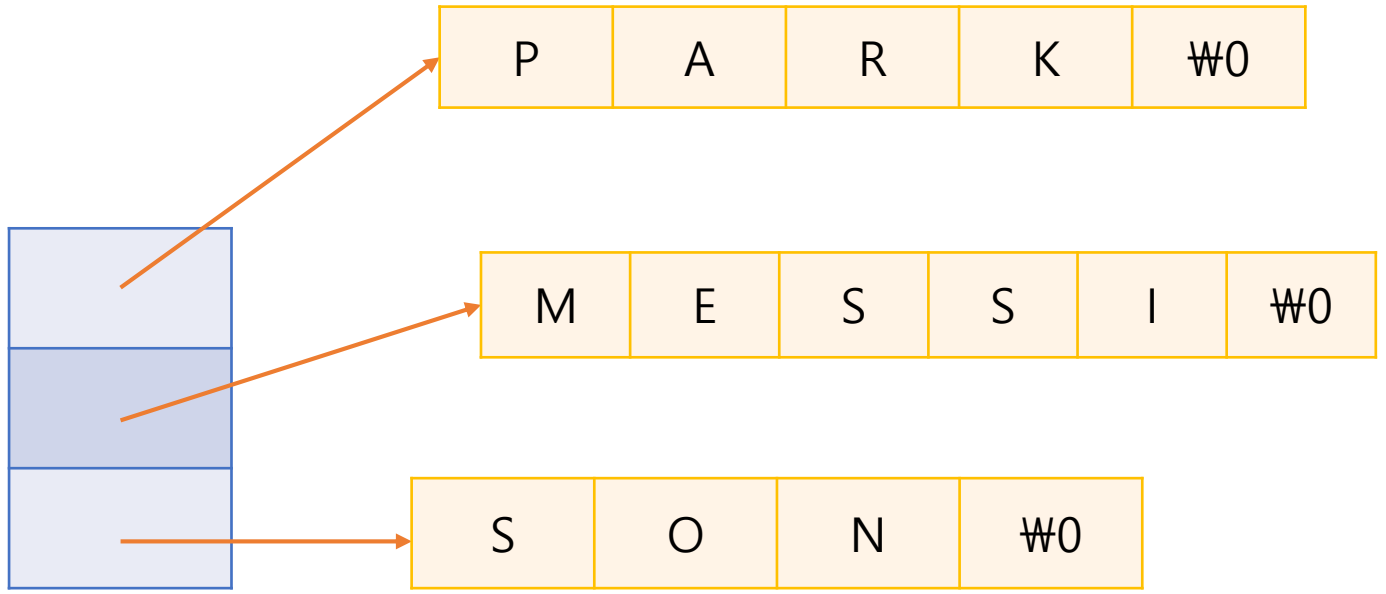
```
int main(void) {

    char names[2][3][6] =
    { { "PARK", "MESSI", "SON" },
      {"KIM", "KANE", "HONG"} };
}
```

| P | A | R | K | ₩0 | |
|---|---|---|---|---|---|
| M | E | S | S | I | ₩0 |
| S | O | N | ₩0 | | |

| K | I | M | ₩0 | | |
|---|---|---|---|---|---|
| K | A | N | E | ₩0 | |
| H | O | N | G | ₩0 | |

# 포인터 배열 (문자열)

```c
int main(void) {

    char* names[3] = { "PARK" , "MESSI", "SON" };

    printf("%s", names[0]);
}
```



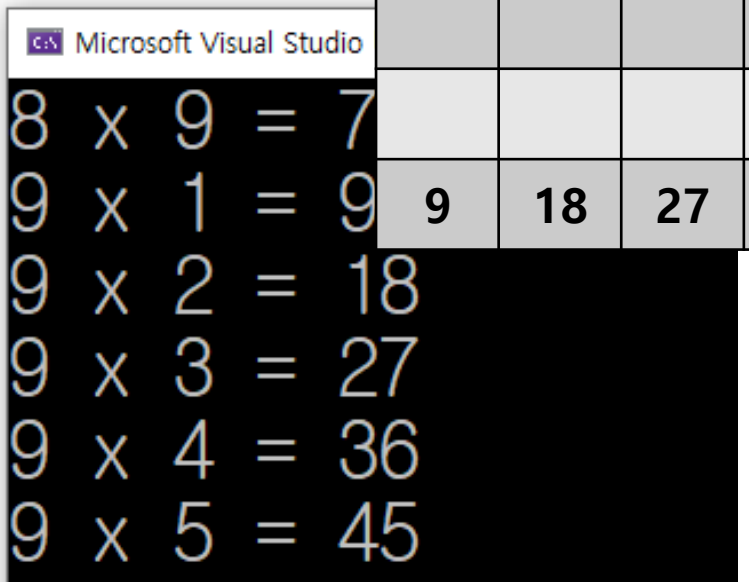
# 다차원 배열 입력 (문자열)

```c
#define MAX_NAME 6

int main(void) {
    char names[3][MAX_NAME];
    puts("1번선수:");
    scanf_s("%s", &names[0], MAX_NAME);
    puts("2번선수:");
    scanf_s("%s", &names[1], MAX_NAME);
    puts("3번선수:");
    scanf_s("%s", &names[2], MAX_NAME);

    printf("%s, %s, %s₩n", names[0], names[1], names[2]);

}
```

| | | | | | |
|---|---|---|---|---|---|
| P | A | R | K | ₩0 | |
| M | E | S | S | I | ₩0 |
| S | O | N | ₩0 | | |

# 다차원 배열에
## 구구단 결과 저장

```
#define MAX_DAN 9
int main(void) {
    int dan[MAX_DAN - 1][MAX_DAN];

    /*
    dan[0][0] = 2 * 1;
    dan[0][1] = 2 * 2;

    dan[1][0] = 3 * 1;
    dan[2][5] = 4 * 6;
    */
```




| 2단 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 9단 |

# 다차원 배열에
## 구구단 결과 저장

```c
#define MAX_DAN 9

int main(void) {
    int dan[MAX_DAN - 1][MAX_DAN];

    for (int i = 0; i < MAX_DAN - 1; i++) {
        for (int k = 0; k < MAX_DAN; k++) {
            dan[i][k] = (i + 2) * (k + 1);
            printf("%d x %d = %d₩n", (i + 2), (k + 1), dan[i][k]);
        }
    }
}
```

# 다차원 배열로
## - 여러 학생 성적 입력

```
int main(void) {

    char names[100][MAX_NAME];
    int scores[100][SCORES];

    int i = 0;
```

```
이름:song
국 영 수 형태로 입력: 39 89 12
--------------------------------

1번 이름 : kim
국어: 20, 영어: 40, 수학: 55
2번 이름 : park
국어: 44, 영어: 95, 수학: 99
3번 이름 : song
국어: 39, 영어: 89, 수학: 12
--------------------------------
```

```
[i] + 2);

));
}
```